Bill Pegram
12/16/2011

Java – C# Differences

Java and C# are very similar but have some differences. The following is an attempt to describe differences in Java and C# concerning topics that would be covered in an introductory semester-length Java programming course such as IT120 at NVCC (data types, operators, selection and repetition statements, methods, arrays, String and Character class methods, and an introduction to object-oriented programming). The discussion of C# is based on *Pro C# 2010 and the .NET 4 Platform*, Fifth Edition, Andrew Troelsen, Apress, 2010, Chapters 3-6. I'm sure this Java – C# comparison is incomplete but hopefully hits things that would be used most frequently.

Java	**C#**
import	using
Java byte code	Common Intermediate Language (CIL)

System.out.println("Hello World");
System.out.print("Hello World");
	Console.WriteLine(("Hello World");
	Console.Write("Hello World");

main method	Main method capitalized
For (elementType element: arrayRefVar)	Foreach (elementType element in arrayRefVar)
Console input with instance methods of the Scanner class – e.g. Scanner input = new Scanner(System.in) String name = input.nextLine);	Console input with static methods of the Console class – e.g. String name = Console.ReadLine();
System.out.printf("Hello %s You are %d years old, userName, userAge);	Console.WriteLine("Hello {0} You are {1} years old, userName, userAge);

Integer.parseInt("8")
Double.parseDouble("8.5")
	int.Parse("8")
	double.Parse("8.5")

"Hello".length()	"Hello".Length caps, no parentheses
all String methods are initial caps after first word, e.g. indexOF	all String methods are initial caps, including the first word
nothing comparable that I'm aware of	verbatim string @"c:\MyApp\bin\debug" (disables processing of escape characters)
nothing comparable that I'm aware of	Implicit typing of local variables (however still results in strongly typed data as opposed to dynamic typing found in languages such as JavaScript) Can be useful with LINQ since you need not explicitly define data type of returned result). ASP.NET 4.0 allows for dynamic typing using keyword dynamic

Only a single value can be returned from a method	Multiple values can be returned from method using the out parameter, e.g.
	Method definition:
	static void Fill(out int a, out int b) {
	a=5;

	b=6;
	}

	Method call:
	static void Main(string[] args) {
	int c, d;
	Fill(out c, out d);
	}
	this will result in c and d after the method call having the values of 5 and 6 respectively

nothing comparable that I’m aware of	Optional parameters (if omitted in call, the specified default values are used), e.g static void Fill(string message, string owner=”Programmer”) {}
Nothing comparable that I’m aware of	Invoking method used Named parameters – can invoke method by specifying parameters is any order, specifying each argument by name using the colon operator, e.g method definition: static void Display(int a, int b, int c) {} method call: Display(3, c: 5, b:6) would pass 6 to b and 5 to c
Nothing comparable that I’m aware of	Array of objects where individual objects are of different types

Constructor chaining using this keyword . e.g.
public Student(String name, int score) {
this.name=name;
this.score=score;}
public Student(int score) {
this(“”, score) ;
optional statements}
	Constructor chaining using this keyword, e.g.
	public Student(string name, int score) {
	this.name=name;
	this.score = score;
	}
	public Student(int score)
	: this(“”,score) {optional statements}

Nothing comparable that I’m aware of	Static constructor – useful when the value is not known at compile time
Nothing comparable that I’m aware of	Static classes
default access modifier for class variables and methods is “protected”	default access modifier for class variables and methods is “internal”

Encapsulation using get/set methods	Can use either
	1) Encapsulation using get/set methods, or
	2) Encapsulation using property syntax, e.g.
	private empName;
	public string Name {
	get {return empName;}
	set {if (value.Length<10)
	empName = value;}
	}
	
	the caller (e.g. from another class) can then write (assuming emp has been

	created as an object of the class)
	emp.Name = “Marv”;
	Console.Write(emp.Name);
	
	as if getting and setting a public piece of data. In the set method, value will have the value “Marv”
	
	The property style is preferred to the separate get/set method style.
	
	static properties are done similarly
	
	Can also use automatic property syntax in which case the private backing field is not specified in the code and object initialization syntax, e.g.
	class Car {
	public string PetName {get; set;}
	public int Speed(get; set;}
	public string Color(get; set; }

final keyword	const keyword (constant fields are by default static
Nothing that I’m aware of	readonly keyword

Nothing that I’m aware of	partial class (pp. 217-218) – class is divided into more than one file, e.g
	one file: class Employee {}
	another file: partial class Employee {}

classical inheritance (is a) public class Minivan extends Car {}	class Minivan: Car
public class final Minivan extends Car {}	sealed class Minivan: Car {}

call base class constructor from derived class constructor (using super) e.g.
public Manager(string fullname, intnumbOpts) {
super(fullname);
stockoptions = numbOpts;}
	call base class constructor from derived class constructor (using super) – e.g.
	public Manager(string fullname, intnumbOpts)
	: base(fullname)
	{stockoptions = numbOpts;}

protected modifier - data can be directly accessed by class in same package or child class of the class	protected modified – data can be directly accessed by child class of the class

Overriding methods – A child class overrides the method in the base class simply by using the same name, with the same signature and return type and the base class method can be invoked by using the keyword super, e.g.
public void GiveBonus() {} in the base class and in the child class
public void GiveBonus() {
super.GiveBonus();
	If a method in the base class can be overridden (but need not be), the base class method uses the keyword virtual and the child class uses the keyword override, e.g. in the base class,
	public virtual void GiveBonus() {}
	and in the child class
	public override void GiveBonus() {
	base.GiveBonus();
	…

…}	} where the latter base refers to the method of the parent class As an alternative (e.g. when you can't change the code of the parent class), you can add the keyword new (pp. 245-247) to hide the implementation of any method with the same name in an ancestor class e.g. public new void GiveBonus() {…} will "override" a GiveBonus() method in an ancestor, even if the latter lacks a virtual keyword